SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE
J. Softw. Maint: Res. Pract. **10**, 333–352 (1998)

Practice

Building Infrastructure for Fixing the Year 2000 Bug: A Case Study

LOUIS J. MARCOCCIA*
MTS/PeopleSource, Selden NY 11784, U.S.A.

SUMMARY

After a brief introduction to the year 2000 (Y2K) software situation, this paper summarizes 10 options for handling the situation, two problems and 16 realities that distinguish Y2K work from other forms of software maintenance. Because it is basic to the case study, this paper then introduces a configuration management system, and briefly explains the 13 objectives for the one used by the organization reported. The Y2K case study coverage begins with a review of the project mission and background about the organization and its information systems environment. The key feature of what this organization did as part of its Y2K project was to build a supporting infrastructure. Of the eight tasks done by the project, only the eighth one was actually devoted to making the organization's software become Y2K compliant. The seven prior infrastructure tasks were: (1) install a single production module, (2) produce system documentation, (3) document all interface data, (4) populate a data dictionary, (5) analyse each production system, (6) produce and use viable JCL, and (7) produce run books. Besides making all the software Y2K compliant within schedule and budget at a cost below the USA national average, the project achieved nine other major benefits. This paper closes with a summary of a generic, successful, Y2K compliance process, and after some discussion, lists five lessons learned. © **1998 John Wiley & Sons, Ltd.**

KEY WORDS: date processing; project management; software inventory; configuration management; data dictionary; maintenance cost

1. INTRODUCTION

The year 2000 adversely affects the performance of a large percentage of existing software (Wessel, 1998; Yardeni, 1998). Most existing software represents the year using a two-character field (e.g., 1998 as 98 and 2000 as 00). This causes date comparisons to give unwanted results. Thus, while people know that the year 2000 comes after the year 1999, does a computer act as though 00 comes after 99? No, a computer regards 00 as smaller than 99, and hence treats 00 as a date as earlier than 99. The software usually adds to the unwanted results by treating a 00 year as representing the year 1900 in particular. This affects calculation of durations in years, depending upon exactly how

* Correspondence to: Louis J. Marcoccia, MTS/PeopleSource, 24 Ambassador Lane, Selden NY 11784–9206, U.S.A. Email: LMarcoccia@aol.com

CCC 1040–550X/98/050333–20$17.50

Received 15 September 1996
Revised 20 February 1998

the data are represented in the computer. For example, the duration between 1955 and 2000 should be 45 years, but a computer may calculate it as –55 years. The duration between 1998 and 2000 should be two years, but a computer may calculate it as 98 years. In general, calculations of durations and any comparisons between dates when one of them is beyond 31 December, 1999, give inaccurate results with most existing information systems. The scope and consequences of this year 2000 date situation have been described in detail by many authors, such as Ulrich and Hayes (1997, pp. 3–42).

Legacy information systems implemented on mainframe computers in COBOL, RPG, 4GL and Assembly programming languages are the most affected by the year 2000 ('Y2K') situation. But nearly all software is suspect until fully tested to assure that it is Y2K compliant (Nandkishore, 1998). This includes software on all sizes and types of platforms, in any language, and running under any operating system or stand alone. Besides application software, it also includes system software (such as operating systems), communications software, packaged software, shareware and even embedded software (Whiting, 1998). Elevator systems, EDI (electronic data interchange) systems, heating and cooling systems, health monitoring systems, systems with software licenses, human resources systems, enterprise-wide systems, and thousands of other kinds of systems may fail to perform as expected because of a lack of Y2K compliance. All Information Technology ('IT') units and all activities using or affected by information systems should review all software they use or by which they are affected. All software-implemented systems are suspect until shown to be Y2K compliant.

The technology to make systems Y2K compliant has existed since the first use of computers for business applications. The technology to avoid the introduction of Y2K non-compliance into software has also existed since the first use of computers for business applications. The reasons that non-compliant systems still exist are the choices that have been made by management over the decades to emphasize short-term benefits and savings over long-term benefits and savings. The attitude has been that if the systems are currently working OK, then leave them alone; and in developing and maintaining systems, act to economize the then–available resources (such as storage space and personnel time). On 16 April, 1996, the present author in testimony at a the U.S. Congressional Hearing pointed out how these choices have created the Y2K situation, and was one of the first persons to testify on the consequences (Marcoccia, 1996). The first person to call public attention to the Y2K situation was William Schoen (Gillin, 1984).

This paper reports on experience gained from making legacy systems Y2K compliant, since that is where the Y2K situation is most critical for most people and organizations. Much of that experience is transferable to other kinds of systems. The process of making systems Y2K compliant, and of assessing the degree of Y2K compliance, relies upon the IT infrastructure, the main focus of this paper. To those ends, this paper looks first at the Y2K problem and some Y2K realities, points out the role and importance of a configuration management system, and then reviews a case study. Afterwards, this paper summarizes a successful Y2K compliance process, and after some discussion, lists five lessons learned.

2. TWO PROBLEMS

There are two main classes of software where the Y2K situation can affect the computer-using organization or person: (1) software used directly by the organization or person, and (2) software used by the person's or organization's customers, and the person's or organizations' suppliers (vendors) and their suppliers, and their suppliers' suppliers, etc. Techniques for fixing the date handling in software have been extensively covered elsewhere (for example, Westbrook (1997), Ulrich and Hayes (1997, pp. 95–132)). For both classes, the 10 main options open for handling the Y2K situation are:

- Expand the databases, the files, the tables and the data descriptions in the software. Some change in the source code is also needed to fit with the date expansion. To save computer storage space when it was much more expensive than it is now, it was common practice to use a two-digit field for the year (YY). Even after a standard date format was agreed upon, many date formats continued to be used such as MM/DD/YY, DD–MM–YY, YYDDD, MMYY, YYMM and YYMMDD. Expansions replace the YY in such formats with YYYY or more rarely CCYY where CC indicates one less than the century except during the last year of a century (recall that century 20 covers the dates from 1 January, 1901 to 31 December, 2000, for example).
- Change the values used to represent dates in the databases, the files, the tables and the data descriptions in the software. This is a technical change that uses the unused part of the binary-coded year representation. A byte can represent 256 symbols; the decimal digits take up only 10 of them. To use such a scheme requires 'overloading' some bit combinations. Many different schemes have been used. All also require changes in the software handling the dates.
- Change the program logic to alter the date representations. The usual way this is done is to put a 'wrapper' around the software to change on input Y2K compliant dates to non-compliant ones, and on output to change non-Y2K compliant dates to compliant ones. Sometimes a limited wrapper or 'mask' can be used to intercept and convert dates (Jones, 1998b, pp. 185–189).
- Change the program logic to alter the way dates are processed. The usual way involves some version of 'windowing' by always treating some low-numbered years as years from 2000 onward, and all other years as years in the 19-hundreds. For example, YY values from 00 to 29 might be treated as indicating the years 2000 to 2029, and values from 30 to 99 as indicating the years 1930 to 1999 (Ouellette, 1997).
- Re–engineer the software along with the databases, files and tables to be Y2K compliant. This involves redesigning and reimplementing the entire system, and changing all of its interfaces with other systems. As a practical matter, this high–risk and relatively expensive process is only done when other changes are also desired in the software.
- Replace the software with software known to be Y2K compliant internally and buffered externally from non-compliant Y2K data. Vendor-supplied software and contractor-written software are the most common choices here, but other sources may be appropriate. Confirming the Y2K compliance of the software can be difficult.

Wrappers or custom-written middleware may be required as a practical matter, when vendor or contractor warranties are deemed weak.

- Retire the software without a replacement. The need for some software has come and gone, yet the software still exists and may even still be run on a regular basis. Retiring the software means physically making the software completely unavailable for running everywhere and by any means. Typically this requires a user sign-off, but is usually the fastest and least expensive way of disposing of Y2K compliance difficulties.
- Practise triage, giving attention to software on a benefit/cost basis, without intending or being able to make all of the software Y2K compliant by when it is needed to be compliant. As 1 January, 2000, gets closer and closer, more and more does triage become the choice (Jones, 1998a; Hayes and Ulrich, 1998). The resources and the time may just not be made available to do much else.
- Hand off the responsibility to someone else or to some other organization to make the software Y2K compliant, as by contracting and outsourcing. The users of the software are the top candidates to take on the responsibility, but the short supply of reasonably-priced capability in the market place to make software Y2K compliant sharply limits the effectiveness of this option.
- Do nothing and/or let the software eventually fail. Unless the IT personnel feel very secure in their positions, this is not a viable alternative. Users of the software are likely to protest and seek penalties or compensation or revenge. Especially when there is no back-up or contingency plan (Anthes, 1998), legal action is a likely possibility, and bankruptcies and business failures are possible (Perdue, 1998; Fenwick, 1997; Draughton, 1998).

These 10 options point to two problems, one psychological and one technical. The technical problem is which option or combination of options to select. That decision depends partly on the existing infrastructure in the organization and the infrastructure that can be built. It also depends upon co-ordinating Y2K-compliance action among the parts of the organization, since some of the output of some systems serves as some of input of other systems, sometimes directly and sometimes indirectly as via a database. Aspects of this technical problem are the focus of this paper.

The psychological problem is a motivational one. Many managers and organization officers still expect a ‘silver bullet’ to become available or that the activities they manage will be unaffected by the Y2K situation. The experience and developments thus far do not provide any objective reliable support for such positions.

As a means of improving personnel recognition of the Y2K situation, the following simple process has sometimes been helpful for personnel who have ready access to a desktop or laptop computer. It takes about 15 minutes. After closing down all running applications, have the person set the date on the computer to 31 December, 1999, and the time to either 11:57 p.m. or to 23:57, and have the person note the actual current time. Then have the person go through the computer shut down procedure, and as soon as it is safe to do so, turn off the power to the computer. While waiting at least five minutes with the computer powered-down, have the person make a list of as many of the items of software and especially the applications that the person runs on the computer

that the person can remember. Then have the person turn on the computer to power it up, and let it attempt to go through its normal start-up procedure. If the start-up fails then either the BIOS or the operating system, or both, are not Y2K compliant, and even compliant applications will not run correctly. If the start up is successful, have the computer display the time and date (Hammond *et al*., 1997). If the time is not correct (not consistent with the change in the actual current time) or the date is not 1 January, 2000, then most date and time processing done within the computer will be inaccurate on all system software and applications. But, if the date and time are correct, that still does *not* mean that each and every one of the items on the list of software and especially applications will perform correctly! Each would have to be tested!

The psychological problem is a significant one. However, since it is not the focus of this paper, it is not covered further here.

3. YEAR 2000 REALITIES

The experience to date reveals the following realities about the Y2K situation that distinguish Y2K compliance work from other software maintenance, and hence about the context in which any supporting infrastructure must operate.

- Most software must be made to be Y2K compliant.
- There is no 'magic pill' or 'silver bullet'; personnel time and effort will be required.
- Costs will be incurred to make software Y2K compliant. In the USA, these costs have been estimated to be around US$200 000 000 000.
- Qualified personnel for Y2K compliance work will be scarce and expensive.
- Contract assistance for Y2K compliance work will be scarce and expensive.
- The only sure benefit from Y2K compliance is avoiding Y2K failures.
- Y2K failures may range from minor 'typos' to becoming bankrupt, depending on the system.
- The final deadline for Y2K compliance cannot be negotiated.
- The proximate deadline for Y2K compliance may be different for different systems.
- Y2K compliance work does not stop when the budget runs out or the deadline is missed.
- Last year is not too soon to start making software Y2K compliant.
- SQA (software quality assurance) and good testing are part of Y2K compliance work.
- A comprehensive configuration management system is required for a controlled change-control process.
- System interface awareness and system integration are part of Y2K compliance work.
- System integration includes working with customers' and suppliers' system interfaces.
- The amount of Y2K compliance work to be done is nearly always underestimated.

For the USA, the Federal Government Internal Revenue Service (IRS) has ruled that the expenditures to do Y2K compliance work are to be treated as current expenses, rather than capitalized and then amortized. This applies even to work on large major systems. Recognized costs typically run between about US$0.20 to US$1.10 and average between about US$0.45 and US$0.70 when done in-house (Kappelman, 1998), and between about

US$1.00 and US$2.75 when contracted or outsourced per line of source code (Jones, 1998b, pp. 79–130; Kappelman *et al.*, 1997). Consider a mission-critical production management system with 15 000 000 lines of source code consisting of just over 7 000 programs (some on–line, some batch) in four languages running under two operating systems on four different platforms at four sites (two overseas), with 265 files accessed by at least two different programs, and accessing seven different major databases using four different database management systems. The initial estimate was that this system had about 200 000 date references. The actual count proved to be nearly 300 000 by the time the Y2K compliance work was completed and tested for a cost of about US$10 000 000.

Missed deadlines, partial completions and inadequate Y2K compliance work is already starting to become a basis for legal action (Draughton, 1998). This trend will continue to grow, but will not peak in terms of the legal expense involved until after 2002 (Perdue, 1998). The legal situation has been likened to that for rear-end collisions of automobiles. The legal liability may arise not just from the actions or inactions of one organization or person, but can be a consequence of the actions or inactions of others, such as of a supplier in a chain of suppliers.

4. CONFIGURATION MANAGEMENT SYSTEM

4.1. Contribution

The Y2K case study described in Section 5 of this paper made extensive use of a configuration management system (Starbuck, 1997). That system, called the Corporate Change-Control Management System (CCM), was used to control the migration of source modules, load modules, job control language (JCL), production procedure statements (PROCS), file and database definitions, and screen maps for each application from test environments to production for all platforms. This process was also intended to provide line-item level change tracking in test or production environments, and was designed to track items in non-source-editable form (e.g., RACF changes and CICS RDO table changes). Applying vendor-supplied updates to packaged software was facilitated by using a distinct ‘language type’ value to identify the package source or object code. To install a new release of the package, the previous version was archived when the new package source was installed, with each of the changes identified and co-ordinated with the aid of the CCM. The CCM also specifically tracked all maintenance of customized software packages.

The CCM was introduced very early in the project described in Section 5 to help achieve a number of objectives. The help came from the features and facilities of the CCM, and from the way the project personnel used the CCM.

4.2. CCM objectives

(1) Consistency. The CCM was to implement and encourage the use of the locally adopted standard procedures and environments through which applications were migrated during development and maintenance. The CCM was to keep information

describing the status of the system and its components in a clear, updated and comprehensible form, and preferably on-line as by using the Endevor PVCS software tool, supplemented by other automated/manual procedures.

(2) Flexibility. The CCM recognized a separate environment for each application group, but the migration rules were consistent for all application groups. Specific environments recognized and applied to all applications were Modification/Test, Development/Test, System Test (optional), User Acceptance Test, Quality Assurance, Production, Training and Post-production Support. The CCM recorded and authorized access to specific files and data ('data sets') for each environment for each application group. Also, the CCM permitted subsets of the environments and procedures to be identified and used.

(3) Audibility. The CCM was to make it possible to trace the history of and the reasons for any changes made, including emergency fixes. The inclusion of test plans, test data files and test run results as trackable items improved the history traces.

(4) Ease and speed of operation. The CCM procedures for effecting migrations were to enable applications to be moved through the required stages of development or maintenance quickly. Although the use of manual forms was not eliminated entirely, migration procedures were automated as far as possible with screens being used for migration specification and authorization. The activities needed to perform a particular migration were easily ascertained from the CCM.

(5) Uniqueness. The CCM helped assure that a given source module existed in only one of the development or maintenance environments at any one time, in addition to the production environment.

(6) Concurrency. The CCM permitted and tracked the emergency repair of a specific module without jeopardizing the concurrent, ongoing, or later maintenance of that module.

(7) Access control. The CCM regulated access to entities and activities to the appropriate authority levels (i.e., only staff assigned to a given applications group could access module types for that group, and only project leaders could initiate forward migrations).

(8) Emergency repair. The CCM provided for a 'Quick Fix' part of the post-production environment into which an application's databases, files, etc., could be migrated following a failure, and into which source code and object code modules could be copied from the production environment for repair and testing. Any concurrent or ongoing maintenance was not affected until the CCM had been notified of the completion of the repair. Monitoring and removal of any overrides placed in the production JCL were the responsibility of the Production Services group.

(9) Recoverability. The CCM provided a means of checking that backout procedures existed to cover cases where an attempted activity was not completed successfully.

(10) Back-up archives. The CCM provided for versioning and for the archiving of prior versions, and made those procedures mandatory for all software used in production. The CCM achieved version uniqueness by carrying out the automated forward migrations in such a way that items were deleted from the originating environment in the event that a migration was successful.

(11) Security. The CCM interacted with the RACF software to provide security and with the CA-Librarian software to provide co-ordination of access control.
(12) Change Log file and Message file. The CCM included a Change Log file for recording all changes to tracked items, and a Message file for maintaining the current migration status of all items.
(13) Audit and management reports. The CCM provided summary activity and status reports, and provided responses to simple *ad hoc* SQL (systems query language) inquiries. The CCM also provided for tracing and reporting audit trails on tracked items.

5. CASE EXAMPLE

5.1. Mission overview

This now-completed three-year project would probably cost nearly US$8 000 000 if it were to have been completed in 1999, but was done within a budget of US$6 000 000. It was for a major organization in New York City using mostly IBM mainframes under the MVS operating system and using COBOL, CICS, IMS, Oracle, SAS and RACF. The project involved making 56 mainframe legacy applications, consisting of about 14 000 000 lines of source code and more than 5 000 jobs, to be Y2K compliant. Costs were kept in line by having the luxury of a three-year working time, by scheduling the project to be done early while qualified personnel were available without a premium cost, and by building (investing in) and using an appropriate supporting infrastructure. While any other project would not be identical to this one, nearly all of the tasks to be done and the situations likely to be encountered are parallel to what was done and encountered in carrying this project to a successful completion within budget and schedule (Delohery and Bucsko, 1997).

An important part of the project might appear to be unrelated to making software Y2K compliant, but as the experience on this project demonstrates, actually is critical to doing the work economically, quickly, accurately and effectively. That important part is building infrastructure to make the existing software more visible and to co-ordinate the changes in the software as the specific Y2K modifications are incorporated. Included in this infrastructure building on this project were:

- establishing a more stable production environment by installing the CCM (a configuration management system as introduced in Section 3);
- cleaning up production jobs without changing their functionality;
- developing and enforcing standards on-line; and
- monitoring the changes on an ongoing basis.

To support the building of the infrastructure, a search was made for a software tool that would generate data flow diagrams and populate a data dictionary, run both on desktop and data centre hardware, and after initial introduction, be usable, without requiring specialist personnel, in the Y2K work and in subsequent routine maintenance. The

IDMS/IDD, an integrated data dictionary tool, was chosen, along with ADW and DOCU-TEXT.

No tools specifically for assisting Y2K compliance work were used on this project. A growing wealth of software tools is becoming available (Olsem and Ragland, 1997; Reps *et al.*, 1997) that could possibly have contributed.

5.2. Background

After an examination of the current mainframe environment by the organization's IT (Information Technology) unit, it was determined that less than five per cent of the production jobs had adequate operating instructions. It was also determined that duplication of software modules in multiple development libraries was extensive, and the manpower needed to support production in the applications was far above the appropriate norm in New York City. Systems, program and operations documentation were all generally obsolete and skimpy, and sometimes non-existent. An analysis of the production update log indicated an extremely low level of program activity. One symptom was of some production jobs being run from development libraries, by non-operations personnel. IT audits of the applications identified 'dirty coding' and latent errors in the software, and pointed out insufficient management control and security (Parker and Stachtchenko, 1996). Of particular concern for Y2K work, the documentation was largely devoid of any trustworthy identification of interface files, and even though software such as CA-Librarian was available, there was no data dictionary implemented serving as a repository for all applications components and data elements.

Interviews with project management highlighted a protracted, round-robin implementation cycle which had forced most of the applications users to circumvent traditionally-accepted production migration routes. The primary users of several major applications insisted that the time had come for either an extensive improvement of their applications, or, a complete rewrite, because of the long lead time necessary to get normal functional enhancements made within these systems. Taken together, these are symptoms of old program code and of insufficient management attention to software maintenance (Chapin, 1993).

5.3. Data centre environment

The organization had a main data centre providing a production environment 24 hours a day 7 days a week, supporting over 20 000 on-line users of terminals and desktop computers throughout the organization. The data centre run applications were processed with the aid of the following hardware and software:

- MVS/XA Operating System running on IBM 3090–300J, IBM 3090–300E, IBM 3084 and IBM 4381 computers;
- IBM 3380 (DASD) providing one terabyte;
- IBM printers, including a 3 800 laser printer and various impact printers;
- IBM 3745 (PEP) for network management;

- AT&T and Timeplex modems and multiplexors, supporting line speeds from 1 200 baud through T–1 and Ethernet, with a Novell LAN; and
- Smaller mainframes and minicomputers throughout the organizations including 35 WANG VS, five IBM Systems 38, an IBM AS400 and one VAX 17.

5.4. Project tasks

5.4.1. Task 1—install a single production module

5.4.1.1. General approach

No attempt was made as a part of this task to get or render an opinion on the quality, effectiveness or suitability of a given program or module with regard to its intended business function in the organization. The work was limited to checking for multiple modules and versions of a program, determining which was the current version and module running in a production or quasi-production status, conducting parallel testing to confirm that said determination was correct, and installing that current version into the data centre production library. To manage that library, the software product CA-Librarian was used. To co-ordinate the work with the client (user) maintenance needs, freeze windows for each application were established and agreed to by IT and the application clients (users). During the freeze window period, the project performed Task 4 work for that application. Once a single production module had been identified, it was necessary to match it to its source code and to place the source code into a protected archive. This ensured that future modifications would be applied to the correct source code.

5.4.1.2. Task 1 deliverables

The deliverable from this task was an undated inventoried manageable production library containing a single version of the production programs for each application. All programs for each application were installed into and run within the data centre's mainframe production environment.

5.4.2. Task 2—produce system documentation for all production software

5.4.2.1. General approach

A set of local standards were developed, published in a local standards manual, and training provided. All applications were then gradually made to conform to the standards with the aid of the ADPAC PM/SS software tool and the documentation tools noted previously (IDMS/IDD, ADW and DOCUTEXT), and the documentation put on-line using an Oracle database after an acceptance approval process.

5.4.2.2. Task 2 deliverables

System documentation:

- narrative description of the system (typically one to three pages);
- an analysis in flow chart form of the system's functional/information processing;
- system outputs for human use (reports and screens layouts with procedures);

- system inputs from hard copy and human beings (documents, keyboarding and screens layouts with procedures); and
- listing of all system interfaces, with sources, destinations and data names.

Models (organization, business, system and data):

- organization models identified which organizational unit and managers used the system, and the name of the responsible manager for the system;
- business models included descriptions of the actual business functions performed by an application;
- system models included the processing cycle, jobs, steps, programs and transactions involved in or affected by each application;
- data models described the logical and physical data formats of databases, files, records, tables and fields for each application; and
- to be approved, all models had to be produced with the aid of the software tools.

5.4.3. Task 3—document all interface data

5.4.3.1. General approach

Each item of data that entered an application from another application or a prior run, or that left an application to go to another application or the same application at a future time, was identified. The name given to the data it was associated with, such as a record name, file name or database name was noted for all applications. This work was an extension and particularization of the work in Task 2.

5.4.3.2. Task 3 deliverables

A report was produced for each system, in control document format, that identified and described all interface records, files, tables and databases existing in all production systems, containing for each record, file, table or database:

- the system and job names that created it;
- the systems(s) and job name(s) and/or foreign designation of it;
- the data set name of it, as applicable;
- the data field content of it;
- the media on which it resided; and
- the timing of its creation.

5.4.4. Task 4—populate a data dictionary/data warehouse

5.4.4.1. General approach

This task also was an extension and particularization of Task 2. All legacy system and client/server data names were rationalized and all sources, definitions and systems fully identified and noted in the integrated data dictionary. Some meta-data items were created when it appeared appropriate. For data used in the legacy systems, a data warehouse was also established from the data dictionary entries.

5.4.4.2. Task 4 deliverables

The two deliverables were the populated IDMS/IDD data dictionary and the data warehouse.

5.4.5. Task 5—analyse each production system and select action

5.4.5.1. General approach

The analysis focused on the current condition of the software implementing the application and any associated or supporting software. The result of the analysis was to assign one of four status levels to the software: major rewrite, partial rewrite, revision or acceptance as is. The analysis identified dead code, code contributing to poor run performance or run instability, and included client (user) views on likely future changes to the system. The analysis did not include evaluating ways to make the software Y2K compliant (see Task 8).

5.4.5.2. Task 5 deliverables

One report deliverable was a set of systems/programs evaluation reports that stated which production applications should undergo major rewriting, partial revision, minor revision or require no revision. Also, recommendations were made about whether certain applications could be re-engineered. The other report deliverable was an oral presentation to the client (user) management; it was required for each system program evaluation report. An additional deliverable was corrected versions of the software with the dead code removed and some run performance or stability improvements made. None of the recommended revisions or rewrites were done as part of any task in this project.

5.4.6. Task 6—produce and use viable JCL and PROCS

5.4.6.1. General approach

This task was limited to production and quasi–production jobs after the completion of Task 5; no development jobs were included. This task required rewriting the JCL to remove all use of non-production files, data sets and program libraries entries, and replace them with standard production naming conventions so that jobs only executed using the production resources for each application. Back-up and restore procedures (PROCS) were developed and incorporated into the JCL for each job step, to assure reliability of applications. Operator overrides and unnecessary condition codes were removed. The IT unit provided an automated diagnostic tool (a 'JCL Filter') for examining any JCL PROCS. This filter identified any non-conformance to JCL standards, reported reason(s) for non-conformance and offending line numbers.

To test the JCL, the Project Manager could have the last production cycle of each application re-run utilizing the changed JCL. If the results were determined by the Project Manager to be reconcilable, the Project Manager accepted the JCL as correct. If the Project Manager did not want to re-execute the last production cycle as an acceptance test, the Project Team examined the changed JCL using methods and tools of their choosing in order to determine the JCL's conformance to the local standard and its performance.

5.4.6.2. Task 6 deliverables

The deliverables were the changed JCL and the changed PROCS to conform to the local standard.

5.4.7. Task 7—produce job operations documentation (run books)

5.4.7.1. General approach

This task was an extension and particularization of Task 2, and was applied to both the production and the quasi-production jobs, as jobs were completed under Task 6. The run book procedures were made following the local IT standard procedures, containing all the data necessary to understand each job's requirements and dependencies, clearly defining external inputs, job requisites, computer resources, outputs by type, job set-up operator procedures, and restore and restart operator procedures for each step.

5.4.7.2. Task 7 deliverables

The deliverables from this task were the job operations documents (run books), that were maintained and used on–line.

5.4.8. Task 8—make the software Y2K compliant

5.4.8.1. General approach

As the CCM took hold and the prior seven tasks were carried to completion, different applications were completed at different times. Completing all seven tasks for an entire application gave the project personnel good visibility into and good change control on that application. This allowed a rapid and accurate analysis of the application for selecting the way of making it Y2K compliant. The good visibility into the cleaned-up applications facilitated making the cost, effort, time and benefit estimates required for doing each of the options listed in Section 2, accommodating the relevant realities listed in Section 3. Interactions among the parts of the application and application group were easily identified, and date data in all forms were easily and accurately recognized.

5.4.8.2. Task 8 deliverables

These were determined on a case by case basis from the CCM by applying the local standards. They included updating the documentation as well as modifying as appropriate the application source code, data descriptions, test plans and test data. The availability of the integrated data dictionary (IDD) and other software tools allowed the production of most deliverables to be automated and often obtained as by-products of producing some other required deliverable.

5.5. Major benefits achieved

(1) Made all applications in all of the application groups and their supporting software become Y2K compliant with no schedule overruns and no budget overruns.
(2) Instituted a configuration management system, the CCM, that has improved the manageability of the organization's investment in information technology.
(3) Developed a enterprise-wide corporate integrated data dictionary. This included rationalizing over one million redundant and alias data elements into 30 000 and documenting them in the data dictionary identifying all sources of data and their uses. This enabled the organization to eliminate duplicate data and provide better security, control and access to its data assets. Meta-data information was also

downloaded to the client/server platform and was later used in a $20 million client/server project involving integrating five of the legacy systems that had been made Y2K compliant.

(4) Centralized production overrides into a single override load library. This allowed data centre operations to have better control in the use of overrides. This enabled operations to place an time limit on how long an override module could be run from an non-production library.
(5) Reallocated data and software among the available storage media to gain improved access times and eliminate inactive data and software. This included reblocking files, which reduced hardware storage costs and speeded up job execution time.
(6) Upgraded all applications. This included putting them into a central controlled production environment for ease of maintenance and security, documenting them to a consistently applied local standard, and improving the run performance of some software.
(7) Created and maintained easily the operational run books, and made them accessible on-line.
(8) Eliminated over 200 non-production software libraries, and the 80 per cent of the existing software modules that were duplicate or incorrect versions and not migrated into production. This saved storage space and improved consistency in meeting the user needs.
(9) Identified and eliminated dead code in all programs, and automated all system-level documentation of the applications.
(10) Facilitated, speeded up and reduced the cost of software maintenance work. This result was unintentional, for the project ran in parallel with, and independent of, all concurrent and ongoing software maintenance. Yet the personnel assigned to maintenance projects had access to and quickly adopted and applied in their work the infrastructure built to support the Y2K compliance work. Software development work also gained, but to a lesser degree.

6. SUCCESSFUL YEAR 2000 COMPLIANCE PROCESS

6.1. Introduction

From the case study and other experience, a general process can be seen for a successful Y2K compliance effort. It has two main phases, analysis and implementation. The purpose of the analysis phase is to estimate the cost and resources needed to implement all application software on diverse software/hardware platforms, including in-house developed and package (third-party vendor and contractor-provided and licensed) software. The purpose of the implementation phase is to implement the tactical plan, application by application, produced from the analysis phase. The lists below are generic and should be augmented and modified to fit specific conditions.

6.2. Analysis phase steps

(1) Strengthen or establish local standards for software, data and documentation.
(2) Strengthen or establish a configuration management system.
(3) Clean up applications to conform to the local standards and bring them within the configuration management system.
(4) Establish a Year 2000 project manager and team.
(5) Gather by questionnaires, to all application users and to the information technology (IT) unit, an application inventory and measure of the potential exposure to Y2K non-compliance.
(6) Establish a date standard for all in-house and contractor-provided software.
(7) Develop an inventory and portfolio of all package software. Formally notify and co-ordinate with the software providers for their schedules to become Y2K compliant.
(8) Develop an inventory of all suppliers to the organization who provide data in computer-accessible form or on documents that are to be computer read. Formally notify and co-ordinate with the suppliers for their schedules to become Y2K compliant.
(9) Have all application users within the organization co-ordinate and agree on one overall strategy, recognizing that the tactical implementations might be different.
(10) Make a preliminary tactical plan for making the software Y2K compliant, and assess the realism of the plan in relation to available management support and available resources.
(11) Prepare impact analyses for the respective application users.
(12) Develop request for proposals (RFP) to select any consulting services by class for all platforms for any services to be outsourced: class 1—turn-key solution; class 2—project management firm; and class 3—time and material contract.
(13) Refine the tactical plan and submit for management and user approval, revising as needed, listing software by user in three categories: 1—software to be converted, 2—software to be replaced; and 3—software that is already Y2K compliant.
(14) After securing management approval and funding, publish the approved plan.

6.3. Analysis phase reports

(1) Impact estimates, including total resources required for the Y2K work, identification of each computer module impacted, total number of impacted lines per program, and total personnel work hours and computer time needed.
(2) Evidence of dates and where they are handled. Where possible (and it will have to be done eventually), locate and print out every impacted line in each impacted program module.
(3) Impact statements for each application's users forecasting what the consequences will be for that application's users if the application does not become fully Y2K compliant.
(4) Project schedule for all category 1 and 2 applications, with an annotated Pert chart or other coverage of the interdependencies among the applications and the stages of the work.

(5) Project Gantt chart for all category 1 and 2 applications.
(6) Cost estimates for category 1 and 2 applications to be done in-house, including all resources such as staff, on-site contractors, hardware, communications, etc., and some major work breakdown, as for example, analyse, change data, change source code and test; and for the category 1 and 2 applications to be outsourced, including conversion as well as any supervision costs, testing costs, contract fees and costs, licence fees and maintenance fees.
(7) Project plan proposal, in coverage and form meeting the organization's usual requirements.

6.4. Implementation phase steps

(1) Secure and mobilize the necessary resources defined in the analysis phase, including the in-house team, the software acquisitions, and the contracted or outsourced services.
(2) Select and do (including testing) a pilot application to verify the cost and schedule model, and after experience with it, make the necessary cost and schedule adjustments based on the pilot results.
(3) Do the rest of the applications and the supporting software in accord with the adjusted plan to make them Y2K compliant.
(4) Resolve the necessary legal/procurement issues involving package software and Y2K compliance clauses in contracts.
(5) Confirm the workability of a back-up plan, and the existence of adequate insurance coverage.
(6) Prepare and pilot test a plan to handle missed Y2K bugs (Myers and Jones, 1997).

6.5. Implementation phase reports

Providing periodic progress and status reports is important. In addition to going to the responsible manager, these reports should also go to the application users. While not part of any formal reporting system, keeping the configuration management system updated should be done very promptly as work is accomplished.

6.6. Helpful software tools

Two critically important tools are a set of software testing tools and a configuration management system, as described in Section 4. An impact analysis tool, a date simulator and a tool to aid in management reporting are also useful (such as a tool to enable responding to *ad hoc* enquiries). The absence of some tools makes Y2K compliance work more difficult, slow and costly, such as the absence of a well-populated integrated data dictionary, and some librarian or software distribution control tool. Specialized Y2K tools can offer assistance, but make a better contribution when applied to well-documented, clean, configuration-managed software.

7. DISCUSSION

At first it may seem odd that infrastructure should have a significant influence on Y2K compliance achievement. Yet in looking at the experiences of many organizations, the same pattern keeps emerging. What work needs to be done, how much of such work needs to be done, and how difficult it will be cannot be reasonably estimated in advance because the organization is ignorant. The organization typically does not know even how many applications it will run in a year's time, let alone how dates are represented, how they are handled in the software, and how many date and date handlings exist and where they exist. This makes impact analysis a guessing game, and cost and schedule estimates politically tinged flights of fancy.

For example, the infrastructure building process revealed how much of the software used by the organization was licenced software. The licence agreements usually limited the kind and amount of changes that could be made, yet the way the licensed software handled and represented dates affected Y2K compliance in other software. A companion difficulty was the software used by the organization's suppliers, since that software generated dates used by the organization's systems. Negotiating with the vendors of the licensed software and with the organization's suppliers was quite different from the usual work of software maintenance, and required co-ordination with the purchasing and legal units within the organization.

Y2K compliance work on in-house developed software can be done quickly, fully and relatively inexpensively when the software and data are clean, consistent and readily visible. This assumes that local standards exist and have been enforced, that a configuration management system is in use and effective, and that as software maintenance has been done, the automated and manual documentation has been kept accurate. The organization does not need to have a CMM maturity rating of five, but it likely could qualify for a maturity rating of two or higher at least with respect to its software maintenance processes (Humphrey, 1989).

Two factors are important about the case study project reported, and are easily overlooked. One is that the project had a three-year planned and actual duration. Since early in 1997, such a planned duration is probably unlikely, as many now current projects will probably find themselves still not finished by 31 December, 1999. The second factor is that the Y2K work was not 'piggybacked' or 'packaged' or combined with other maintenance work other than the elimination of all dead code and some instances of run performance improvement. That is, the Task 8 Y2K work was a distinct task, not commingled with any other software maintenance work, even when other maintenance work on the same software happened to be being done concurrently, as was the case in a few instances.

What can an organization do when those happy states of affairs do not exist, or when time is running short? As the case study reported here indicates, it may be worthwhile to build the infrastructure to accomplish a clean-up in at least a specific application area before attempting to take on the nuts and bolts of the actual Y2K compliance work in that area. Because of the way records were kept and the tasks done during the case study project, the Task 8 work could not be clearly separated and was not specifically measured.

The informal feeling was that less than half of the total project cost and effort went into Task 8, given that Tasks 1 to 7 were accomplished.

Drawing upon the experience reported in the case study and upon experience with other Y2K client organizations, this paper has laid out a set of necessary steps needed to build the infrastructure and plan required for a successful Y2K project. The infrastructure building includes a clean up process before the actual Y2K compliance work is begun. Depending upon conditions, all or a portion of the infrastructure building process must be done to create a foundation on which the constant changes that will occur in an organization's systems can be handled quickly and effectively as normal software maintenance. Then, making software Y2K compliant becomes just another change. To make that possible, a solid infrastructure must be in place to ensure the greatest chance of being successful in achieving Y2K compliance—because organizations cannot afford to do such software maintenance poorly.

8. LESSONS LEARNED

The organization learned five lessons from the Y2K compliance effort reported in the case study in this paper.

(1) An organization with typical legacy systems, weak documentation, so-so operations and diffuse user community can make its mission-critical systems be year 2000 (Y2K) compliant at a cost below average, and gain additional benefits as by-products.
(2) Making in-house developed software be year 2000 complaint is just another software maintenance project, but making licensed and vendor or third-party (package) software year 2000 compliant is both a software maintenance project and a supplier-relations project.
(3) Building infrastructure to facilitate the year 2000 software maintenance work made the year 2000 compliance work proceed faster and at a lower cost, and also speeded other software maintenance work and reduced its costs, giving a lasting improvement for the organization.
(4) Relying on a limited selection of software tools and associated procedures made the process of building infrastructure go smoothly. The principal tools used were a configuration management system, an integrated data dictionary, a software library system and a set of documentation tools. A year 2000 compliance project can be carried to a successful conclusion without specific year 2000 software tools when the software is clean and well documented.
(5) Establishing and enforcing a set of local standards bridged over gaps in communication, sharply reduced exceptions, reduced the burden of understanding the software, and on net balance, reclaimed some hardware capacity. The enforced standards made the tools and associated procedures more effective, and made the year 2000 compliance work go more quickly.

References

Anthes, G. H. (1998) 'When disaster strikes', *Computerworld*, **32**(3), 80–81, 83.

Chapin, N. (1993) 'Software maintenance characteristics and effective management', *Journal of Software Maintenance*, **5**(2), 91–100.

Delohery, P. D. and Bucsko, J. (1997) 'A blessing in disguise', *Datamation*, **43**(10), 29–30.

Draughton, R. S. (1998) 'Year 2000 litigation update', *Information Executive*, **2**(2), 4 and **2**(4), 5.

Fenwick, W. A. (1997) *The Year 2000 Problem Legal Issues and Suggestions to Address Them*, Fenwick & West LLP, Palo Alto CA, 19 pp.

Gillin, P. (1984) 'The problem you may not know you have', *Computerworld*, **18**(6), 7.

Hammond, E., Carreon, J. C., Go, A. and McClure, S. (1997) 'Debacle on the desktop', *Infoworld*, **19**(45), 106–107, 110.

Hayes, I. S. and Ulrich, W. M. (1998) *The Year 2000 Software Crisis: Challenge of the Century*, Prentice Hall PTR, Upper Saddle River NJ, 454 pp.

Humphrey, W. S. (1989) *Managing the Software Process*, Addison-Wesley Publishing Co., Reading MA, 494 pp.

Jones, T. C. (1998a) 'Getting down to the wire', *Application Development Trends*, **5**(4), 37–38, 40, 44, 46.

Jones, T. C. (1998b) *The Year 2000 Software Problem*, Addison Wesley Longman, Inc., Reading MA, 335 pp.

Kappelman, L. A. (1998) 'Progress report', *Software Magazine Year 2000 Survival Guide*, **18**(6), 29–30.

Kappelman, L. A., Fent, D., Keeling, K. B. and Prybutok, V. R. (1997) 'Calculating the cost of year–2000 compliance', *Communications of the ACM*, **41**(2), 30–39.

Marcoccia, L. J. (1996) 'Year 2000', in *January 1, 2000, the Date for Computer Disaster*, a hearing before the Subcommittee on Government Management, Information, and Technology, of the Committee on Government Report and Oversight, House of Representatives, available as document ISBN 0–16–055416–0, Superintendent of Documents, Washington DC, pp. 39–44, 64–67.

Myers, W. and Jones, T. C. (1997) 'Slow response to year 2000 problem', *IEEE Software*, **14**(3), 14–15.

Nandkishore, N. (1998) 'The next step: validating year 2000 compliance', *Application Development Trends*, **5**(3), 43–44, 46, 48.

Olsem, M. R. and Ragland, B. (1997) 'USAF year 2000 tool evaluation project', *Cross*Talk, **10**(10), 7; available as http://www.stsc.hill.af.mil/crosstalk/1997/oct/y2ktool.html.

Ouellette, T. (1997) 'Insurer "lies" to avoid year 2000 costs', *Computerworld*, **31**(3), 1, 16.

Perdue, L. (1998) 'A staggering problem', *Silicon Valley Tech Week*, **1**(4), 1, 18–19.

Parker, R. G. and Stachtchenko, P. (1996) *Year 2000 Readiness Kit*, Information Systems Audit and Control Association, Rolling Meadows IL, 8 pp.

Reps, T., Ball, T., Das, M. and Larus, J. (1997) 'The use of program profiling for software maintenance with applications to the year 2000 problem', *Software Engineering Notes*, **22**(6), 432–449.

Starbuck, R. A. (1997) 'Software configuration management: don't buy a tool first', *Cross*Talk, **10**(11), 28–30.

Ulrich, W. M. and Hayes, I. S. (1997) *The Year 2000 Software Crisis: The Continuing Challenge*, Prentice Hall PTR, Upper Saddle River NJ, 587 pp.

Wessel, D. (1998) 'Year 2000 is costly but not catastrophic', *Wall Street Journal*, **138**(86), A1.

Westbrook, D. E. (1997) 'Programming alternatives and repair actions for the year 2000,' in Kappelman, L. A. (Ed), *Year 2000 Problem: Strategies and Solutions from the Fortune 100*, International Thomson Computer Press, Boston MA, pp. 114–119.

Whiting, R. (1998) 'The embedded legacy', *Software Magazine Year 2000 Survival Guide*, **18**(6), 23–24, 26.

Yardeni, E. (1998) 'Y2K—an alarmist view', *Wall Street Journal*, **138**(86), A22.

Author's biography:

Louis J. Marcoccia is a Year 2000 Consultant and Principal with MTS/PeopleSource. He has been working on the year 2000 situation in software since 1991. He has planned and directed entire year 2000 projects for diverse organizations, especially in government, insurance and finance, and consulted on other such projects. Also he has planned, directed and consulted on parts of year 2000 projects and related projects, such as on software packaging, restart and recovery, change control, code conversion, legacy system clean up, testing, and standards and procedures. Lou has conducted two-day workshops on year 2000 compliance, been a speaker on year 2000 topics at national meetings, and given testimony on the topic at U. S. Congressional Hearings. His email address is: LMarcoccia@aol.com